Introduction to the M-file Connexions Modules

MATLAB has emerged as a widely used computational tool in many fields of engineering. MATLAB consists of a programming language used in an interactive computing environment that supports the development of programs to solve complex problems. The MATLAB language has become a defacto standard that is also used by several other computational packages, including LabVIEW MathScript and Octave. Generically, we refer to these packages as **m-file environments** because the program files typically are identified by an extension of "m".

The Connexions modules in this course are intended to introduce freshman engineering students to problem solving using an m-file environment. Most of the information in these modules applies to any m-file environment (MATLAB, LabVIEW MathScript, Octave, etc.). There are some differences between environments, and occasionally some material will be specific to a given environment. This material is offset from the surrounding text and labeled with the appropriate environment. For example:

Note: Matlab is a commercial product of The MathWorks.

Note: LabVIEW MathScript is a commercial product of National Instruments.

Note: Octave is an open source environment that is available without charge. Information about Octave is available at the Octave home page.

Using MATLAB

Matlab Help

MATLAB has a great on-line help system accessible using the help command. Typing

```
help <function>
```

will return text information about the chosen function. For example to get information about the built-in function sum type:

```
help sum
```

To list the contents of a toolbox type help <toolbox>, e.g. to show all the functions of the signal processing toolbox enter

```
help signal processing
```

If you don't know the name of the function but a suitable keyword use the `lookfor` followed by a keyword string, e.g.

```
lookfor 'discrete fourier'
```

To explore the extensive help system use the "Help menu" or try the commands `helpdesk` or `demo`.

Matrices, vectors and scalars

MATLAB uses matrices as the basic variable type. Scalars and vectors are special cases of matrices having size 1x1, 1xN or Nx1. In MATLAB, there are a few conventions for entering data:

- Elements of a row are separated with blanks or commas.
- Each row is ended by a semicolon, ;.
- A list of elements must be surrounded by square brackets, [ ]

Example:
It is easy to create basic variables.
`x = 1` (scalar)
`y = [2 4 6 8 10]` (row vector)
`z = [2; 4; 6; 8; 10]` (column vector)
`A = [4 3 2 1 0; 1 3 5 7 9]` (2 x 5 matrix)

Regularly spaced values of a vector can be entered using the following compact notation

`start:skip:end`

Example:
A more compact way of entering variables than in Example 1 is shown here:
`y= 2 : 2 : 10`
`A=[4:-1:0;1:2:9]`

If the skip is omitted it will be set to 1, i.e., the following are equivalent

`start:1:end` and `start:end`

To create a string use the single quotation mark " ' ", e.g. by entering `x = 'This is a string'`.

Indexing matrices and vectors

Indexing variables is straightforward. Given a matrix M the element in the i'th row, j'th column is given by `M(i,j)`. For a vector v the i'th element is given by `v(i)`. Note that the lowest allowed index in MATLAB is 1. This is in contrast with many other programming languages (e.g. JAVA and C), as well as the common notation used in signal processing, where indexing starts at 0. The colon operator is also of great help when accessing specific parts of matrices and vectors, as shown below.

Example:
This example shows the use of the colon operator for indexing matrices and vectors.
`A(1,:)` returns the first row of the matrix A.
`A(:,3)` returns the third column of the matrix A.
`A(2,1:5)` returns the first five elements of the second row.
`x(1:2:10)` returns the first five odd-indexed elements of the vector x.

Basic operations

MATLAB has built-in functions for a number of arithmetic operations and functions. Most of them are straightforward to use. The Table below lists the some commonly used functions. Let x and y be scalars, M and N matrices.

	MATLAB
xy	`x*y`
x^y	`x^y`
e^x	`exp(x)`
$\log(x)$	`log10(x)`

	MATLAB
$\ln(x)$	`log(x)`
$\log2(x)$	`log2(x)`
MN	`M*N`
M^{-1}	`inv(M)`
M^T	`M'`
$\det(M)$	`det(M)`

Common mathematical operations in MATLAB

- Dimensions - MATLAB functions length and size are used to find the dimensions of vectors and matrices, respectively.
- Elementwise operations - If an arithmetic operation should be done on each component in a vector (or matrix), rather than on the vector (matrix) itself, then the operator should be preceded by ".", e.g .*, .^ and ./.

Example:
Elementwise operations, part I
Let $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. Then `A^2` will return $\mathrm{AA} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$, while `A.^2` will return $\begin{pmatrix} 1^2 & 1^2 \\ 1^2 & 1^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

Example:
Elementwise operations, part II

Given a vector x, and a vector y having elements $y(n) = \frac{1}{\sin(x(n))}$. This can be easily be done in MATLAB by typing `y=1./sin(x)` Note that using `/` in place of `./` would result in the (common) error `Matrix dimensions must agree`.

Complex numbers

MATLAB has excellent support for complex numbers with several built-in functions available. The imaginary unit is denoted by `i` or (as preferred in electrical engineering) `j`. To create complex variables $z_1 = 7 + i$ and $z_2 = 2e^{i\pi}$ simply enter `z1 = 7 + j` and `z2 = 2*exp(j*pi)`

The Table below gives an overview of the basic functions for manipulating complex numbers, where z is a complex number.

	MATLAB
$\text{Re}(z)$	`real(z)`
$\text{Im}(z)$	`imag(z)`
$\|z\|$	`abs(z)`

	MATLAB
Angle(z)	`angle(z)`
z^*	`conj(z)`

Manipulating complex numbers in MATLAB

Other Useful Details

- A **semicolon** added at the end of a line tells MATLAB to suppress the command output to the display.
- MATLAB and **case sensitivity**. For variables MATLAB is case sensitive, i.e., b and B are different. For functions it is case insensitive, i.e., sum and SUM refer to the same function.
- Often it is useful to **split a statement** over multiple lines. To split a statement across multiple lines, enter three periods `"..."` at the end of the line to indicate that it continues on the next line.

Example:
Splitting $y = a + b + c$ over multiple lines. `y = a... + b... + c;`

Graphical representation of data in MATLAB

Graphical representation of data in MATLAB

MATLAB provides a great variety of functions and techniques for graphical display of data. The flexibility and ease of use of MATLAB's plotting tools is one of its key strengths. In MATLAB graphs are shown in a figure window. Several figure windows can be displayed simultaneously, but only one is active. All graphing commands are applied to the active figure. The command `figure(n)`will activate figure number `n` or create a new figure indexed by `n`.

Tools for plotting

In this section we present some of the most commonly used functions for plotting in MATLAB.

- `plot`- The plot and stem functions can take a large number of arguments, see help plot and help stem. For example the line type and color can easily be changed. `plot(y)` plots the values in vector `y`versus their index. `plot(x,y)` plots the values in vector `y`versus `x`. The `plot` function produces a piecewise linear graph between its data values. With enough data points it looks continuous.
- `stem`- Using `stem(y)`the data sequence `y`is plotted as stems from the x-axis terminated with circles for the data values. `stem` is the natural way of plotting sequences. `stem(x,y)` plots the data sequence `y` at the values specified in `x`.
- `xlabel('string')`- Labels the x-axis with `string`.
- `ylabel('string')`- Labels the y-axis with `string`.
- `title('string')`- Gives the plot the title `string`.

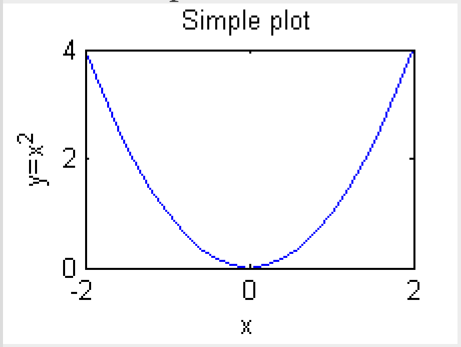
To illustrate this consider the following example.

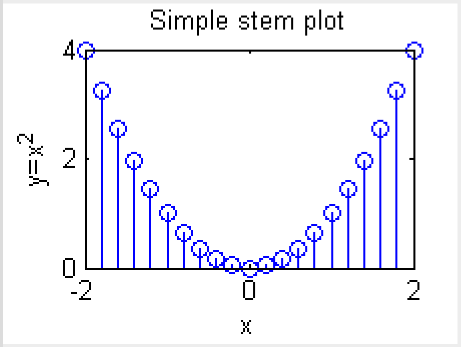
Example:

In this example we plot the function y = x2 for x 2 [-2; 2].

```
x = -2:0.2:2;
y = x.^2;
figure(1);
plot(x,y);
xlabel('x');
ylabel('y=x^2');
title('Simple plot');
figure(2);
stem(x,y);
xlabel('x');
ylabel('y=x^2');
title('Simple stem plot');
```

This code produces the following two figures.





Some more commands that can be helpful when working with plots:

- hold on / off - Normally hold is off. This means that the plot command replaces the current plot with the new one. To add a new plot to an existing graph use `hold on`. If you want to overwrite the current plot again, use `hold off`.

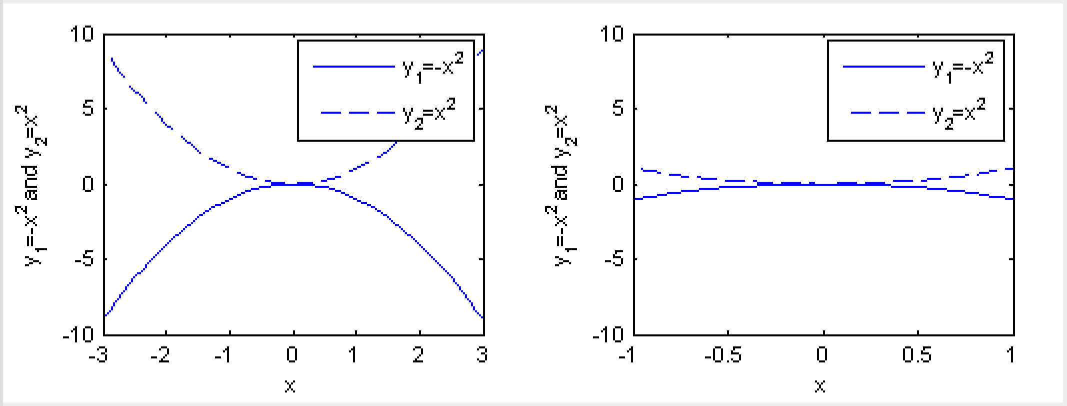
- `legend('plot1','plot2',...,'plot N')`- The `legend` command provides an easy way to identify individual plots when there are more than one per figure. A legend box will be added with strings matched to the plots.
- `axis([xmin xmax ymin ymax])`- Use the `axis` command to set the axis as you wish. Use `axis on/off` to toggle the axis on and off respectively.
- `subplot(m,n,p)` -Divides the figure window into `m` rows, `n` columns and selects the `p`p'th subplot as the current plot, e.g `subplot(2,1,1)` divides the figure in two and selects the upper part. `subplot(2,1,2)` selects the lower part.
- `grid on/off` - This command adds or removes a rectangular grid to your plot.

Example:
This example illustrates `hold`, `legend` and `axis`.

```
x = -3:0.1:3; y1 = -x.^2; y2 = x.^2;
figure(1);
plot(x,y1);
hold on;
plot(x,y2,'--');
hold off;
xlabel('x');
ylabel('y_1=-x^2 and y_2=x^2');
legend('y_1=-x^2','y_2=x^2');
figure(2);
plot(x,y1);
hold on;
plot(x,y2,'--');
hold off;
xlabel('x');
ylabel('y_1=-x^2 and y_2=x^2');
```

```
legend('y_1=-x^2','y_2=x^2');
axis([-1 1 -10 10]);
```

The result is shown below.

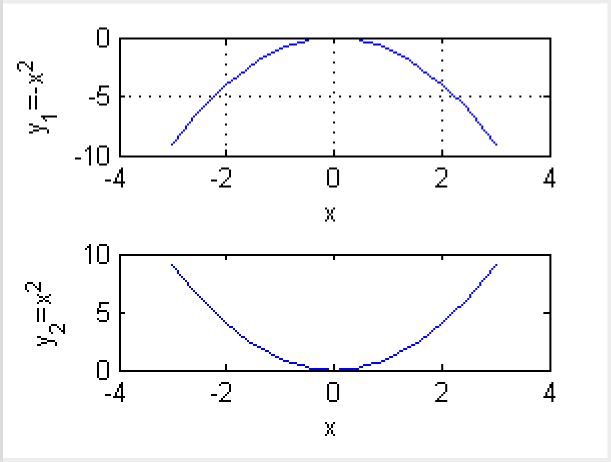


Example:

In this example we illustrate subplot and grid.

```
x = -3:0.2:3; y1 = -x.^2; y2 = x.^2;
subplot(2,1,1);
plot(x,y1);
xlabel('x'); ylabel('y_1=-x^2');
grid on;
subplot(2,1,2);
plot(x,y2);
xlabel('x');
ylabel('y_2=x^2');
```

Now, the result is shown below.



Printing and exporting graphics

After you have created your figures you may want to print them or export them to graphic files. In the "File" menu use "Print" to print the figure or "Save As" to save your figure to one of the many available graphics formats. Using these options should be sufficient in most cases, but there are also a large number of adjustments available by using "Export setup", "Page Setup" and "Print Setup".

To streamline the graphics exportation, take a look at exportfig package at Mathworks.com, URL: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=727.

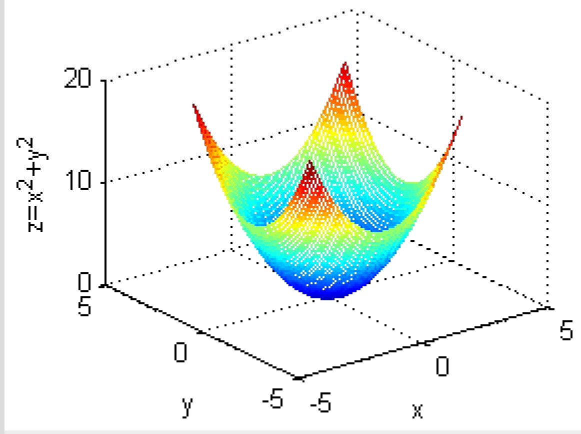
3D Graphics

We end this module on graphics with a sneak peek into 3D plots. The new functions here are `meshgrid` and `mesh`. In the example below we see that `meshgrid`produces `x`and `y`vectors suitable for 3D plotting and that `mesh(x,y,z)` plots `z` as a function of both `x` and `y`.

Example:
Example: Creating our first 3D plot.

```
[x,y] = meshgrid(-3:.1:3);
z = x.^2+y.^2;
mesh(x,y,z);
xlabel('x');
ylabel('y');
zlabel('z=x^2+y^2');
```

This code gives us the following 3D plot.



z=x^2+y^2
20
10
0
5
0
-5
y
-5
0
5
x